Wyszukiwarka semantyczna z bazą opartą na Wikipedii (wybrana tematyka)

Radosław Bujak
Michał Borowski
Michał Gątkowski

Cel projektu:

W ramach projektu należało skonstruować wyszukiwarkę semantyczną z bazą opartą na Wikipedii. Rozwiązanie polegało na wybraniu artykułów z danej tematyki, a następnie zastosowaniu różnych algorytmów konstrukcji wektorów słów w celu stworzenia wyszukiwarki semantycznej zdolnej do wyszukiwania artykułów z Wikipedii na podstawie zapytania.

Wykorzystane technologie:

Projekt zaimplementowano przy użycia języka programowania *Python* wraz z kilkoma bibliotekami wspomagającymi przetwarzanie języka naturalnego.

Do przygotowania danych wykorzystano następujące rozwiązania:

- https://en.wikipedia.org/wiki/Special%3AExport - rozwiązanie wikipedii pozwalające na wyeksportowanie artykułow o podanych tytułach do formatu *xml*.
- https://github.com/attardi/wikiextractor - skrypt pozwalający na wstępne przetworzenie tekstu artykułów pobranych z wikipedii w formacie *xml*. *WikiExtractor* usuwa zbędne znaki wynikające z notacji zapisu Wikipedii. W rezultacie otrzymujemy pliki zawierające informacje o tytule, id, url, tekście artykułu w przystępnym formacie *xml*.
- *Simple api for xml* - jest to wbudowana biblioteka pythona do operowania na plikach *xml*.
- *Natural language toolkit* - platforma wspomagająca budowanie aplikacji rozwiązujących problemy języka naturalnego w języku *Python.*

Do stworzenia wyszukiwarki semantycznej wykorzystano następujące rozwiązania:

- Biblioteka *NumPy* wspomagająca przeprowadzane w ramach projektu obliczenia.
- Biblioteka *scikit-learn* zawierająca gotowe implementacje algorytmów Data Science. Wykorzystano z niej m.in gotowe implementacje *PCA, LDiA, SVD, TF-IDF* etc.

Do stworzenia GUI wykorzystano bibliotekę *Dear PyGui*, która pozwala na tworzenie prostego, dynamicznego, wspomaganego przez GPU graficznego interfejsu użytkownika.

Dane treningowe

Dane treningowe pochodzą z angielskiej wikipedii z wszystkich kategorii powiązanych z przetwarzaniem danych, komputerologią (ang. *computing)*. W skład tej tematyki znalazło się 29 głównych kategorii z czego otrzymano 2250 artykułów (po przetworzeniu ich końcowo wykorzystano 2237 artykułów). Artykuły te niestety nie pokrywają się z wszystkimi artykułami związanymi z tym tematem. Wynika to z działania wikipedii, gdzie nie każdy wpis ma przypisaną kategorię. Samo określenie wszystkich rzeczywistych wpisów wikipedii na dany temat mogłoby stanowić spory problem (nie zajmowano się tym w projekcie).

Po zidentyfikowaniu artykułów wykonano kilka niezbędnych kroków w celu przygotowania ich do wykorzystania przy budowie wyszukiwarki semantycznej:

1. Pobrano przy pomocy narzędzia wikipedii *Special Export* wszystkie artykuły do formatu *xml*.
2. Wstępnie przetworzono tekst artykułów przy pomocy skryptu *WikiExtractor,* aby pozbyć się notacji wikipedii.
3. Przygotowano tekst artykułów, aby nadawały się do zbudowania wektorów słów:
    a. Usunięto znaki interpunkcyjne oraz inne nic nie wnoszące znaki.
    b. Ztokenizowano tekst artykułu.
    c. Zamieniono duże litery na małe.
    d. Usunięto słowa nieinformatywne.
    e. Testowano również stemowanie oraz lemantyzację, ale w końcowej wersji zrezygnowano z tego etapu ze względu na gorsze wyniki.
4. W ostatnim etapie przygotowania danych wyeksportowano każdy artykuł do osobnego pliku tekstowego, który zawierał informacje o id, tytule, url artykułu oraz ztokenizowany tekst (tokenized word1, tokenized word2 etc.).

Implementacja

Program został napisany jako aplikacja okienkowa. Klasa *SemanticSearch* odpowiada za logikę programu. Do jej konstruktora przekazywane są:

- *corpus* - list artykułów, gdzie każdy artykuł składa się z listy przetworzonych słów
- *urls* - lista linków do danych artykułów
- topics - lista tytułów artykułów
- *distance_type* - metryka obliczania odległości pomiędzy wektorami
- *vectorizer* - typ algorytmu tworzącego wektor słów
- decomposer - typ algorytmu redukującego wymiary macierzy słów
- *n_components* - liczba komponentów, parametr dla *decomposer'a*
- *normalization* - opcja pozwalająca przeprowadzić normalizację na zadanej macierzy słów przed redukcją wymiarowości

Metryka odległości pomiędzy wektorami

Do obliczania odległości wykorzystano funkcję *pairwise_distance* z biblioteki *scikit-learn.* Użytkownik może wybrać jedną z trzech metryk: *l1, l2* oraz *cosin*.

Wektor słów

Do stworzenia wektorów słów wykorzystano dwie metody: *TF-IDF* oraz *Bag of Words* zaimplementowane w bibliotece *scikit-learn*.

Redukcja wymiarowości

Przy redukcji wymiarowości wykorzystano 3 algorytmy: *SVD*, *PCA* oraz *LDiA* również zaimplementowane w bibliotece *scikit-learn*.

Dla zadanych algorytmów program dopasowuje się i przekształca do podanych danych w liście *corpus* w metodzie *prepare_model*

```python
def prepare_model(self):
    corpus_vectorized = self.vectorizer.fit_transform(self.corpus).toarray()
    if self.normalization:
        corpus_vectorized = self.scaler.fit_transform(corpus_vectorized)
    return self.decomposer.fit_transform(corpus_vectorized)
```

Rys. 1 *Fragment kodu prezentujący przygotowywanie modelu.*

Do przeprowadzenia normalizacji wykorzystano klasę *MinMaxScaler()* z biblioteki *scikit-learn*.

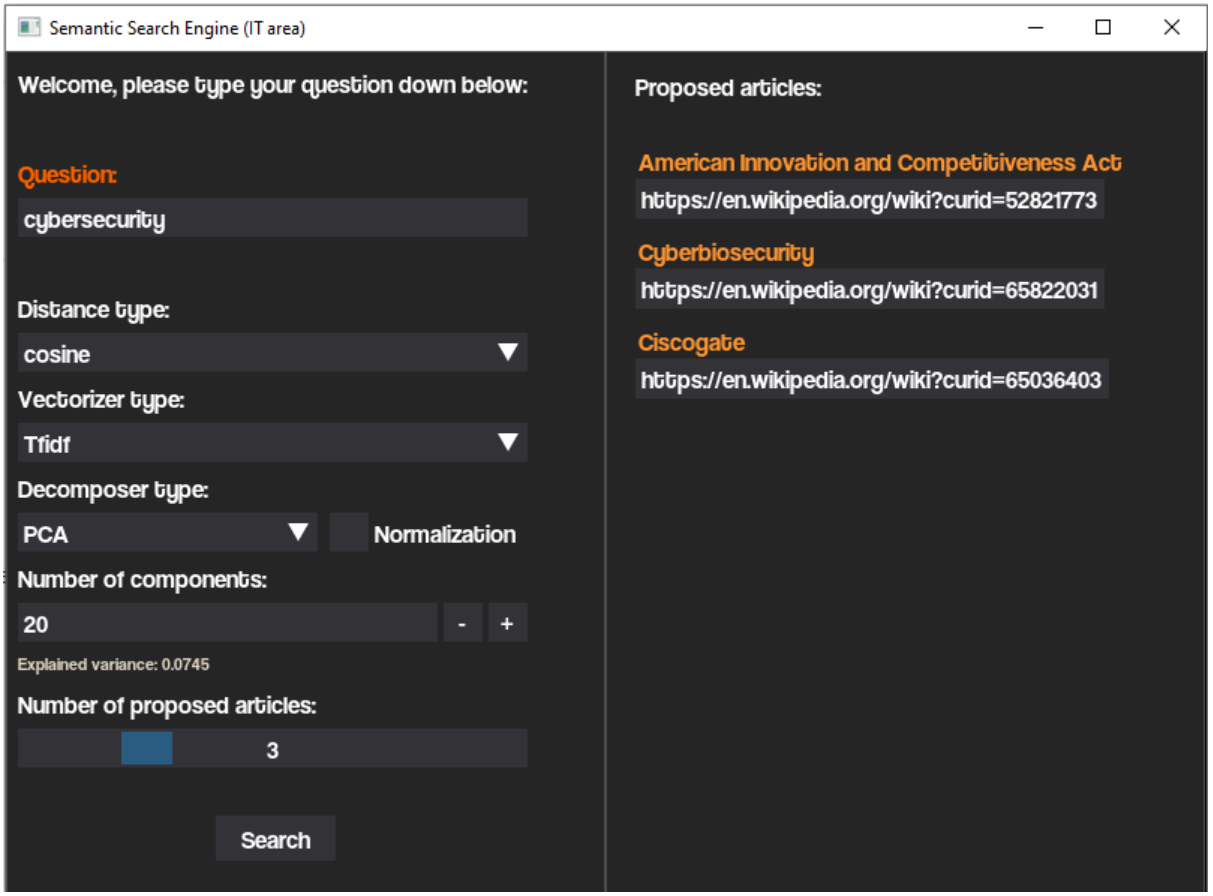
Po przygotowaniu modelu program jest gotowy na otrzymanie zapytania od użytkownika. Początkowo pytanie jest przekształcane przy pomocy algorytmu do tworzenia wektora słów. Następnie obliczane są odległości o danej metryce pomiędzy wektorami w modelu, a pytaniem. Im mniejsza odległość tym lepsze dopasowanie pytania do artykułu. Po obliczeniu odległości wybierane jest *n* najlepszych. Za tę część logiki odpowiada funkcja *search*:

```python
def search(self, question, n_articles) -> list:
    transformed_question = self.transform_question(question.lower())
    distances = self.get_distances(self.model, transformed_question)
    best_indexes = self.get_n_min_indexes(distances, n_articles)
    responses = []
    for index in best_indexes:
        responses.append((self.topics[index], self.urls[index]))
    return responses
```

Rys. 2 *Fragment kodu prezentujący funkcję zwracającą najlepiej dopasowane artykuły do pytania użytkownika.*

Interfejs

Za wyświetlanie interfejsu odpowiada klasa SemanticSearchGUI wykorzystująca bibliotekę *dearpygui*. Po uruchomieniu programu wyświetla się pokazane poniżej okno:



Rys. 3 *Interfejs aplikacji*

Po lewej stronie okienka użytkownik wpisuje pytanie (w polu *your question*), a następnie wybiera dostępne opcje modelu. Użytkownik może wybrać:

- metrykę do obliczania dystansu pomiędzy wektorami (*Distance type*)
- algorytm tworzenia wektora słów (*Vectorizer type*)
- algorytm redukcji wymiarowości (*Decomposer type*)
- czy wektor słów ma być znormalizowany przed redukcją wymiarowości (*Normalization*)
- liczbę komponentów dla danego algorytmu redukcji wymiarowości (*Number of components*). Minimalna wartość to 1, a maksymalna to liczba artykułów użyta podczas przygotowywania modelu. Dla zadanej liczby komponentów obliczana jest również wariancja wyjaśniona, która wyświetla się poniżej tego pola. Nie jest ona dostępna dla algorytmu LDiA.
- liczba proponowanych artykułów (*Number of proposed articles*)

Po prawej stronie znajduje się wyjście programu, czyli tytuł oraz linki z odnośnikami dla najlepiej dopasowanych artykułów dla danego modelu. Po kliknięciu w link użytkownik przenoszony jest do odpowiedniego artykułu na Wikipedii.

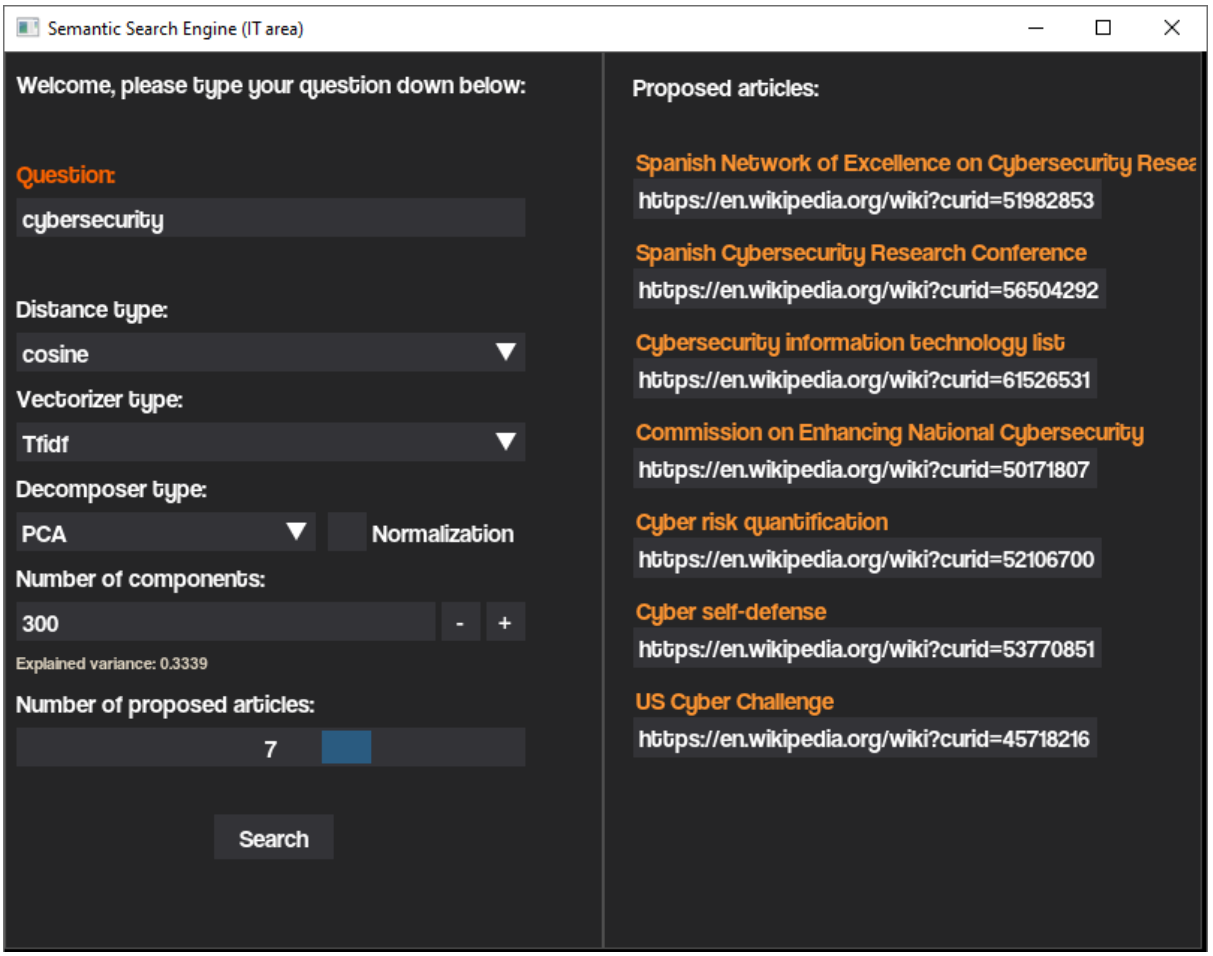
Uruchomienie aplikacji

Aplikację można uruchomić w wybranym IDE lub w konsoli, będąc w folderze projektu, przy użyciu komendy:

python main.py

Testowanie aplikacji

Po przetestowaniu kilku ogólnych słów okazuje się, że program trafnie wyszukuje odpowiednie artykuły powiązane z tematem. Poniżej przykład dla słowa *cybersecurity*:



Rys. 4 *Przykład działania aplikacji dla zapytania "cybersecurity".*

Wpisując w google zapytanie "*best companies ai wikipedia*" otrzymuje się artykuły, których część znajduje się w naszym zbiorze testowym. Aplikacje testowano czy wyszukując tę samą frazę w stworzonej wyszukiwarce semantycznej otrzymamy podobne wyniki.

```
AI Companies of India
List of artificial intelligence projects
Artificial intelligence
Applications of artificial intelligence
OpenAI
Artificial intelligence industry in China
```

Rys. 5 *Artykuły pokrywające się z danymi testowymi aplikacji po wyszukaniu w google zapytania "best ai companies".*

Testy wyboru różnych metryk do obliczania odległości pomiędzy wektorami nie wykazały dużych różnic między wyszukanymi artykułami. Inaczej sytuacja miała się w przypadku wyboru algorytmu tworzenia wektora słów. *Bag of Words* dawał zupełnie inne artykuły, dużo mniej trafne, niż *TF-IDF*. Również zastosowanie normalizacji spowodowało wyszukiwanie gorszych artykułów. Zdecydowano się więc na porównanie pod kątem algorytmu do redukcji wymiarowości oraz liczby komponentów bez normalizacji z użyciem *TF-IDF*.

Porównanie działania aplikacji pod kątem wyboru algorytmu do redukcji wymiarowości (dekompozycja) i liczbę wymiarów (komponentów).

Najpierw porównano czy liczba artykułów wyszukiwanych przez google(rys. 5), a znajdujących się w zbiorze testowym różni się w zależności od wykorzystanego algorytmu do redukcji wymiarowości i liczby wymiarów (jest łącznie 6 wspólnych artykułów). Na podstawie tabeli 1, widać, że najlepiej radzącymi sobie algorytmami do redukcji wymiarowości są *PCA* i *SVD* otrzymujące od 1 do 3 trafionych artykułów w zależności od liczby wymiarów. Można zauważyć również, że wraz z większą ilością wymiarów wyniki stają się dokładniejsze. Wynika to prawdopodobnie z faktu traconych informacji przy redukcji do niewielkiej liczby wymiarów.

Tabela przedstawiająca liczbę artykułów pokrywających się z wyszukiwaniem google w zależności od liczby wymiarów i algorytmu redukcji wymiarowości dla wektora słów TF-IDF wraz z mierzoną długością przy pomocy L2			
Typ algorytmu do redukcji wymiarowości/Ilość wymiarów	**5**	**50**	**500**
PCA	1	2	3
SVD	1	2	3
LDiA	0	0	0

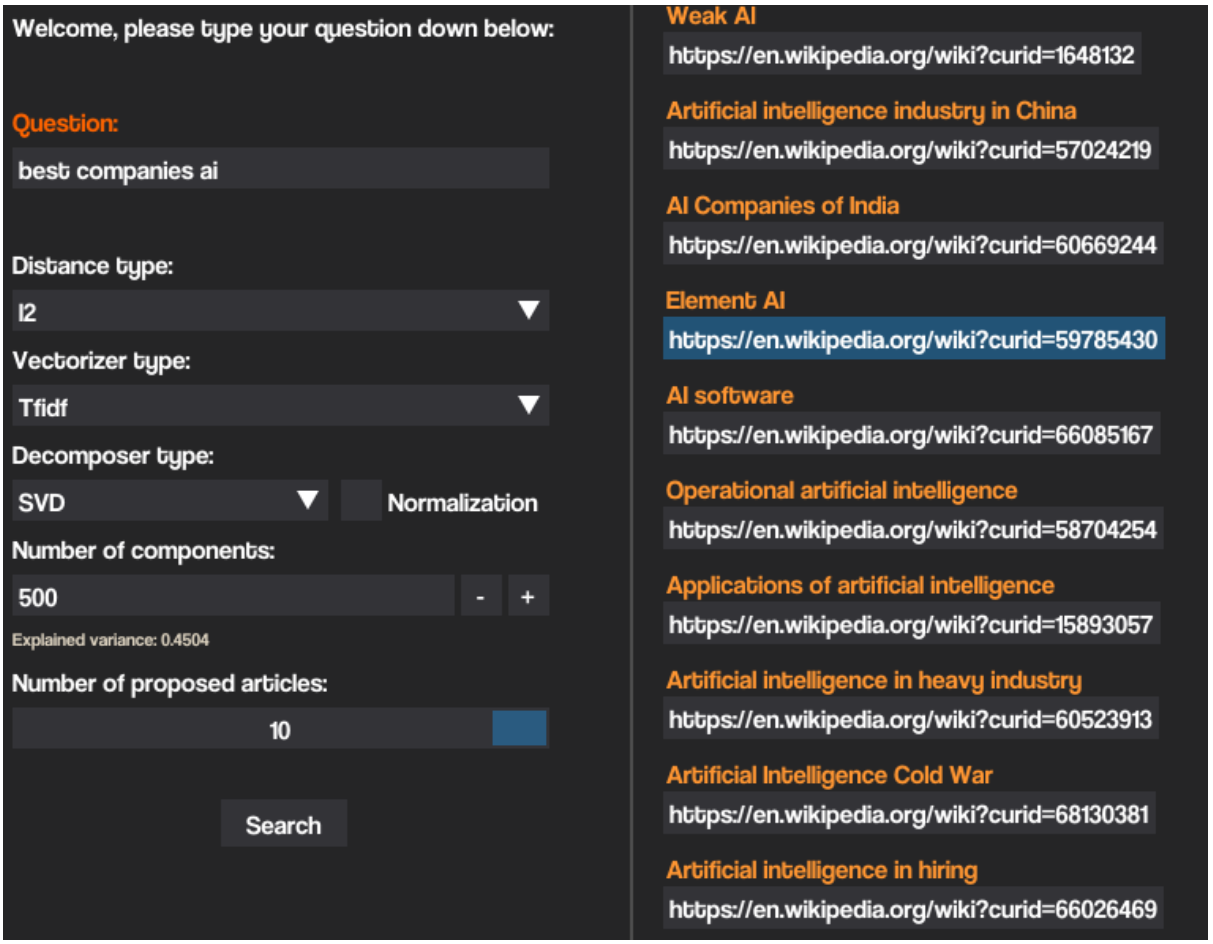
Tabela 1 *Tabela przedstawiająca liczbę artykułów pokrywających się z wyszukiwaniem google w zależności od liczby wymiarów i algorytmu redukcji wymiarowości dla wektora słów TF-IDF wraz z mierzoną długością przy pomocy L2*

Bazując na samym zapytaniu *“best companies ai”* i zrozumieniu artykułów znajdujących się w zbiorze testowym najwyższą pozycję powinny zajmować artykuły *“AI Companies of India”* oraz “*Artificial Intelligence industry in china*”. Przygotowano, więc tabelę analizującą pozycje tych artykułów w zależności od algorytmu do redukcji wymiarowości i ilości wymiarów. Na podstawie tabeli 2 widać, że różnica w pozycji artykułów pomiędzy *PCA, SVD* jest nieznaczna i dla podanego zapytania nie widać specjalnych różnic pomiędzy nimi.

Tabela przedstawiająca pozycję wybranych najważniejszych artykułów w zależności od liczby wymiarów i algorytmu redukcji wymiarowości dla wektora słów TF-IDF wraz z mierzoną długością przy pomocy L2			
Typ algorytmu do redukcji wymiarowości/Ilość wymiarów	**5**	**50**	**500**
PCA	(9, None)	(1,5)	(3,2)
SVD	(8, None)	(1,6)	(3,2)
LDiA	(None,None)	(None,None)	(None,None)

Tabela 2 *Tabela przedstawiająca pozycję wybranych najważniejszych artykułów w zależności od liczby wymiarów i algorytmu redukcji wymiarowości dla wektora słów TF-IDF wraz z mierzoną długością przy pomocy L2*

Na poniższym rysunku zamieszczono konfigurację uzyskującą możliwie najlepsze wyniki dla zapytania "*best companies ai*".



Rys. 6 *Screen aplikacji zawierający najlepszą testowaną konfigurację aplikacji.*

Podsumowanie

W ramach projektu udało się zrealizować wyszukiwarkę semantyczną w oparciu o wybraną tematykę wikipedii (wybrano tematykę *computing*). Projekt składał się z 4 najważniejszych etapów:

1. Przygotowanie danych: pobranie artykułów z wikipedii o wybranej tematyce oraz przygotowanie ich, aby nadawały się do tworzenia wektorów słów.
2. Implementacja semantycznej wyszukiwarki: wykorzystanie biblioteki *scikit-learn* do implementacji: metryk do obliczania dystansu, algorytmów tworzenia wektorów słów oraz algorytmów redukcji wymiarowości.
3. Stworzenie interfejsu graficznego aplikacji.
4. Testowanie aplikacji.

Porównując wykorzystane algorytmy najlepszym okazał się być *SVD* oraz *PCA,* które otrzymywały bardzo podobne wyniki (jedyną przewagą był czas tworzenia modelu na korzyść *SVD*). Wykorzystane do testowania zapytanie "*best companies ai*" uzyskało podobne wyniki w google ("*best companies ai wikipedia*") co jest bardzo obiecującym wynikiem. Z wybranego zbioru testowego google, było w stanie wyszukać 6 artykułów, natomiast nasza aplikacja maksymalnie 3 artykuły.